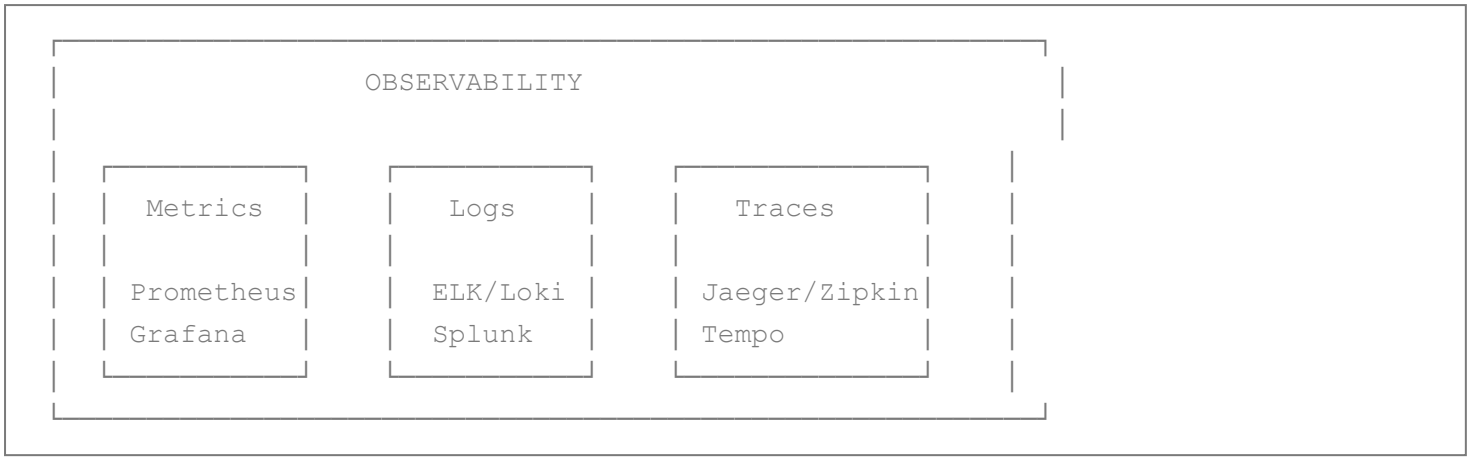
Observability & Security

Version: 1.0
Date: 2025-11-26
Status: Implementation Ready

---

1. Observability Strategy

1.1 Three Pillars



---

2. Logging Strategy

2.1 Structured JSON Logging

```
<!-- logback-spring.xml -->
<configuration>
    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder class="net.logstash.logback.encoder.LogstashEncoder">
            <includeMdcKeyName>traceId</includeMdcKeyName>
            <includeMdcKeyName>spanId</includeMdcKeyName>
            <includeMdcKeyName>signatureId</includeMdcKeyName>
            <includeMdcKeyName>customerId</includeMdcKeyName>
            <includeMdcKeyName>provider</includeMdcKeyName>
            <includeMdcKeyName>channelType</includeMdcKeyName>

            <customFields>
                {"service":"signature-router","env":"${SPRING_PROFILES_ACTIVE}"}
            </customFields>
        </encoder>
    </appender>

    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
```

```
        <file>logs/signature-router.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>logs/signature-router.%d{yyyy-MM-
dd}.%i.log.gz</fileNamePattern>
            <maxFileSize>100MB</maxFileSize>
            <maxHistory>30</maxHistory>
            <totalSizeCap>10GB</totalSizeCap>
        </rollingPolicy>
        <encoder class="net.logstash.logback.encoder.LogstashEncoder"/>
    </appender>

    <logger name="com.bank.signature" level="INFO"/>
    <logger name="org.springframework" level="WARN"/>
    <logger name="org.hibernate" level="WARN"/>

    <root level="INFO">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="FILE"/>
    </root>
</configuration>
```

2.2 MDC (Mapped Diagnostic Context)

```
@Component
@Order(0)
public class MdcFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(
        HttpServletRequest request,
        HttpServletResponse response,
        FilterChain filterChain
    ) throws ServletException, IOException {

        try {
            // Generate or extract trace ID
            String traceId = request.getHeader("X-Trace-Id");
            if (traceId == null) {
                traceId = UUID.randomUUID().toString();
            }

            MDC.put("traceId", traceId);
            MDC.put("path", request.getRequestURI());
            MDC.put("method", request.getMethod());
            MDC.put("clientIp", getClientIp(request));

            // Add trace ID to response header
```

```
            response.setHeader("X-Trace-Id", traceId);

            filterChain.doFilter(request, response);
        } finally {
            MDC.clear();
        }
    }

    private String getClientIp(HttpServletRequest request) {
        String ip = request.getHeader("X-Forwarded-For");
        if (ip == null || ip.isEmpty()) {
            ip = request.getRemoteAddr();
        }
        return ip;
    }
}
```

2.3 Domain-Specific MDC

```
@Service
public class StartSignatureUseCaseImpl implements StartSignatureUseCase {

    @Override
    public SignatureResponse start(CreateSignatureRequest dto) {
        SignatureRequest request = SignatureRequest.create(dto.getCustomerId(),
context);

        // Add domain context to MDC
        MDC.put("signatureId", request.getId().toString());
        MDC.put("customerId", tokenize(request.getCustomerId()));  // Tokenized,
no PII

        try {
            // Business logic...
            log.info("Signature request created: signatureId={}, customerId={}",
                request.getId(), tokenize(request.getCustomerId()));

            return SignatureMapper.toResponse(request);
        } finally {
            MDC.remove("signatureId");
            MDC.remove("customerId");
        }
    }

    private String tokenize(String customerId) {
        // Already pseudonymized in DB, but double-check
        return customerId.substring(0, 8) + "...";  // Log only prefix
```

```
    }
}
```

2.4 Sample Log Output

```
{
  "@timestamp": "2025-11-26T10:30:01.234Z",
  "level": "INFO",
  "service": "signature-router",
  "env": "production",
  "traceId": "550e8400-e29b-41d4-a716-446655440000",
  "spanId": "a1b2c3d4",
  "signatureId": "018c7f5e-1234-7890-abcd-ef1234567890",
  "customerId": "CUST_8f7a...",
  "provider": "TWILIO",
  "channelType": "SMS",
  "path": "/api/v1/signatures",
  "method": "POST",
  "clientIp": "192.168.1.100",
  "message": "Challenge sent successfully",
  "duration_ms": 145,
  "thread": "http-nio-8080-exec-1"
}
```

No PII in logs: `customerId` tokenized, no phone numbers, emails, or personal data.

---

3. Metrics (Prometheus + Micrometer)

3.1 Business Metrics

```
@Component
public class SignatureMetrics {

    private final Counter signatureCreated;
    private final Counter signatureCompleted;
    private final Counter signatureExpired;
    private final Counter challengeSent;
    private final Counter challengeFailed;
    private final Timer signatureDuration;
    private final Gauge activeSignatures;

    public SignatureMetrics(MeterRegistry registry) {
        this.signatureCreated = Counter.builder("signature.created")
            .description("Total signature requests created")
            .tags("service", "signature-router")
            .register(registry);
```

```
        this.signatureCompleted = Counter.builder("signature.completed")
            .description("Total signatures completed")
            .tags("service", "signature-router")
            .register(registry);

        this.signatureExpired = Counter.builder("signature.expired")
            .description("Total signatures expired")
            .tags("service", "signature-router")
            .register(registry);

        this.challengeSent = Counter.builder("challenge.sent")
            .description("Challenges sent by channel")
            .tag("service", "signature-router")
            .register(registry);

        this.challengeFailed = Counter.builder("challenge.failed")
            .description("Challenges failed by provider")
            .tag("service", "signature-router")
            .register(registry);

        this.signatureDuration = Timer.builder("signature.duration")
            .description("Time from creation to completion")
            .tags("service", "signature-router")
            .publishPercentiles(0.5, 0.95, 0.99)  // P50, P95, P99
            .register(registry);

        this.activeSignatures = Gauge.builder("signature.active",
                () -> getActiveSignatureCount())
            .description("Number of active signature requests")
            .register(registry);
    }

    public void recordSignatureCreated() {
        signatureCreated.increment();
    }

    public void recordChallengeSent(ChannelType channel, String provider) {
        challengeSent.increment(Tag.of("channel", channel.name()),
                                Tag.of("provider", provider));
    }

    public void recordChallengeFailed(ChannelType channel, String provider, String
errorCode) {
        challengeFailed.increment(
            Tag.of("channel", channel.name()),
            Tag.of("provider", provider),
```

```
                Tag.of("errorCode", errorCode)
            );
        }

        public void recordSignatureCompleted(Duration duration, ChannelType
finalChannel, int attempts) {
            signatureCompleted.increment(
                Tag.of("finalChannel", finalChannel.name()),
                Tag.of("attempts", String.valueOf(attempts))
            );

            signatureDuration.record(duration);
        }
    }
```

3.2 Provider Metrics

```
    @Component
    public class ProviderMetrics {

        private final Counter providerCalls;
        private final Counter providerErrors;
        private final Timer providerLatency;
        private final Gauge providerErrorRate;

        public ProviderMetrics(MeterRegistry registry) {
            this.providerCalls = Counter.builder("provider.calls")
                .description("Total calls to providers")
                .register(registry);

            this.providerErrors = Counter.builder("provider.errors")
                .description("Total provider errors")
                .register(registry);

            this.providerLatency = Timer.builder("provider.latency")
                .description("Provider API latency")
                .publishPercentiles(0.5, 0.95, 0.99)
                .register(registry);

            this.providerErrorRate = Gauge.builder("provider.error_rate",
                    this::calculateProviderErrorRate)
                .description("Provider error rate percentage")
                .register(registry);
        }

        public void recordProviderCall(String provider, Duration latency, boolean
success) {
```

```
        providerCalls.increment(Tag.of("provider", provider));
        providerLatency.record(latency, Tag.of("provider", provider));

        if (!success) {
            providerErrors.increment(Tag.of("provider", provider));
        }
    }
}
```

3.3 Fallback Metrics

```
@Component
public class FallbackMetrics {

    private final Counter fallbackAttempts;
    private final Counter fallbackSuccess;
    private final Counter fallbackExhausted;

    public FallbackMetrics(MeterRegistry registry) {
        this.fallbackAttempts = Counter.builder("fallback.attempts")
            .description("Fallback attempts by channel")
            .register(registry);

        this.fallbackSuccess = Counter.builder("fallback.success")
            .description("Successful fallbacks")
            .register(registry);

        this.fallbackExhausted = Counter.builder("fallback.exhausted")
            .description("Fallback chain exhausted")
            .register(registry);
    }

    public void recordFallbackAttempt(ChannelType from, ChannelType to) {
        fallbackAttempts.increment(
            Tag.of("from", from.name()),
            Tag.of("to", to.name())
        );
    }
}
```

3.4 Prometheus Endpoint

```yaml
management:
  endpoints:
    web:
      exposure:
        include: health,info,metrics,prometheus
  metrics:
    export:
      prometheus:
        enabled: true
    tags:
      application: signature-router
      environment: ${SPRING_PROFILES_ACTIVE}
```

Prometheus Scrape Config:

```yaml
# prometheus.yml
scrape_configs:
  - job_name: 'signature-router'
    metrics_path: '/actuator/prometheus'
    scrape_interval: 15s
    static_configs:
      - targets: ['signature-router:8080']
```

---

4. Distributed Tracing

4.1 Micrometer Tracing Configuration

```yaml
management:
  tracing:
    sampling:
      probability: 1.0  # 100% sampling (adjust in production)

  zipkin:
    tracing:
      endpoint: http://zipkin:9411/api/v2/spans

spring:
  sleuth:
    enabled: true
    sampler:
      probability: 1.0
```

4.2 Trace Propagation

```
@Component
public class SignatureProviderAdapter implements SignatureProviderPort {

    private final Tracer tracer;

    @Override
    public ProviderResult sendChallenge(SignatureChallenge challenge) {
        // Create child span
        Span span = tracer.nextSpan().name("provider.sendChallenge");

        try (Tracer.SpanInScope ws = tracer.withSpan(span.start())) {
            span.tag("provider", challenge.getProvider().name());
            span.tag("channel", challenge.getChannelType().name());
            span.tag("signatureId", challenge.getSignatureRequestId().toString());

            // Propagate trace context to provider HTTP call
            ProviderResult result = twilioClient.send(challenge);

            span.tag("result", result.isSuccess() ? "success" : "failure");

            return result;
        } catch (Exception ex) {
            span.error(ex);
            throw ex;
        } finally {
            span.end();
        }
    }
}
```

4.3 Kafka Trace Propagation

```
@Service
public class OutboxEventPublisher implements EventPublisherPort {

    private final Tracer tracer;
    private final KafkaTemplate<String, GenericRecord> kafkaTemplate;

    @Override
    public void publish(DomainEvent event) {
        Span span = tracer.currentSpan();

        // Create Kafka message with trace context
        ProducerRecord<String, GenericRecord> record = new ProducerRecord<>(
            "signature.events",
```

```
            event.getAggregateId(),
            eventToAvro(event)
        );

        // Inject trace context into Kafka headers
        if (span != null) {
            record.headers().add("X-B3-TraceId",
span.context().traceId().getBytes());
            record.headers().add("X-B3-SpanId",
span.context().spanId().getBytes());
        }

        kafkaTemplate.send(record);
    }
}
```

4.4 Sample Trace

```
Trace ID: 550e8400-e29b-41d4-a716-446655440000

┌─ POST /api/v1/signatures [145ms]
│  ├─ StartSignatureUseCase.start [140ms]
│  │  ├─ RoutingService.evaluateRoute [5ms]
│  │  ├─ SignatureRequestRepository.save [15ms]
│  │  ├─ provider.sendChallenge [110ms]
│  │  │  ├─ TwilioSmsProvider.send [105ms]
│  │  │  │  └─ HTTP POST api.twilio.com [100ms]
│  │  │  └─ ProviderMetrics.record [5ms]
│  │  └─ EventPublisher.publish [10ms]
│  └─ SignatureMapper.toResponse [5ms]
```

---

5. Dashboards (Grafana)

5.1 SLO Dashboard

```
{
  "dashboard": {
    "title": "Signature Router - SLOs",
    "panels": [
      {
        "title": "P99 Latency (Target: <300ms)",
        "targets": [
          {
            "expr": "histogram_quantile(0.99, signature_duration_bucket)"
          }
        ],
```

```
        "alert": {
          "conditions": [
            {
              "evaluator": {
                "type": "gt",
                "params": [300]
              }
            }
          ]
        }
      },
      {
        "title": "Availability (Target: ≥99.9%)",
        "targets": [
          {
            "expr": "sum(rate(signature_created[5m])) -
sum(rate(signature_failed[5m])) / sum(rate(signature_created[5m])) * 100"
          }
        ]
      },
      {
        "title": "Error Rate (Target: <0.1%)",
        "targets": [
          {
            "expr": "sum(rate(signature_failed[5m])) /
sum(rate(signature_created[5m])) * 100"
          }
        ]
      }
    ]
  }
}
```

5.2 Business Metrics Dashboard

- Signatures created per minute
- Signatures completed per minute
- Average time to completion
- Fallback rate (%)
- Cost savings (Push vs SMS)

5.3 Provider Health Dashboard

- Provider calls per minute
- Provider error rate (%)
- Provider latency (P50, P95, P99)
- Circuit breaker status
- Degraded mode timeline

---

6. Alerting Rules

6.1 Critical Alerts

```
# alertmanager.yml
groups:
  - name: signature_router_critical
    interval: 30s
    rules:
      - alert: HighP99Latency
        expr: histogram_quantile(0.99, signature_duration_bucket) > 300
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "P99 latency exceeds SLO (>300ms)"
          description: "P99 latency is {{ $value }}ms"

      - alert: HighErrorRate
        expr: sum(rate(signature_failed[5m])) / sum(rate(signature_created[5m])) >
0.001
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Error rate exceeds 0.1%"

      - alert: AllProvidersDown
        expr: sum(provider_degraded_mode) == count(provider_degraded_mode)
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "All signature providers are down"

      - alert: DatabaseConnectionPool Exhausted
        expr: hikaricp_connections_active >= hikaricp_connections_max
```

```
        for: 2m
        labels:
          severity: critical
```

6.2 Warning Alerts

```
  - name: signature_router_warnings
    interval: 1m
    rules:
      - alert: ProviderDegradedMode
        expr: provider_degraded_mode == 1
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Provider {{ $labels.provider }} in degraded mode"

      - alert: HighFallbackRate
        expr: sum(rate(fallback_attempts[5m])) / sum(rate(signature_created[5m]))
> 0.1
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "Fallback rate exceeds 10%"

      - alert: CircuitBreakerOpen
        expr: resilience4j_circuitbreaker_state{state="open"} == 1
        for: 2m
        labels:
          severity: warning
```

---

7. Security

7.1 Authentication & Authorization

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())  // API, use tokens
            .authorizeHttpRequests(auth -> auth
```

```
                .requestMatchers("/actuator/health", "/actuator/info").permitAll()
                .requestMatchers("/api/v1/signatures/**").authenticated()
                .requestMatchers("/api/v1/admin/**").hasRole("ADMIN")
                .anyRequest().authenticated()
            )
            .oauth2ResourceServer(oauth2 -> oauth2
                .jwt(jwt -> jwt.jwtAuthenticationConverter(jwtConverter()))
            )
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            );

        return http.build();
    }

    @Bean
    public JwtAuthenticationConverter jwtConverter() {
        JwtAuthenticationConverter converter = new JwtAuthenticationConverter();
        converter.setJwtGrantedAuthoritiesConverter(jwt -> {
            List<String> roles = jwt.getClaimAsStringList("roles");
            return roles.stream()
                .map(role -> new SimpleGrantedAuthority("ROLE_" + role))
                .collect(Collectors.toList());
        });
        return converter;
    }
}
```

7.2 Roles & Permissions

Role	Permissions
`USER`	Create signatures, view own signatures
`ADMIN`	All USER permissions + manage rules, view all signatures
`AUDITOR`	Read-only access to audit logs and metrics
`SUPPORT`	Read-only access to signatures (without PII)

7.3 Pseudonymization Service

```
@Service
public class PseudonymizationService {

    private final SecretManagerPort secretManager;

    public String pseudonymize(String realCustomerId) {
        // Use keyed HMAC for deterministic pseudonymization
        String key = secretManager.getSecret("pseudonymization-key");

        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(key.getBytes(), "HmacSHA256"));

        byte[] hash = mac.doFinal(realCustomerId.getBytes());
        return "CUST_" + Base64.getUrlEncoder().encodeToString(hash)
            .substring(0, 16);  // 16 chars + prefix
    }

    // Note: Pseudonymization is ONE-WAY (no de-pseudonymization in this system)
    // Real customer data lives in separate Customer Service with strict access
control
}
```

7.4 Secrets Management (Vault)

```
spring:
  cloud:
    vault:
      uri: https://vault.bank.com:8200
      authentication: KUBERNETES  # or TOKEN, APPROLE
      kubernetes:
        role: signature-router
        service-account-token-file:
/var/run/secrets/kubernetes.io/serviceaccount/token
      kv:
        enabled: true
        backend: secret
        default-context: signature-router
        profiles:
          - prod
```

```
@Component
public class VaultSecretManager implements SecretManagerPort {

    private final VaultTemplate vaultTemplate;
```

```
    @Override
    public String getProviderCredential(String provider) {
        VaultResponse response = vaultTemplate.read(
            "secret/signature-router/" + provider
        );

        if (response == null || response.getData() == null) {
            throw new IllegalStateException("Secret not found for provider: " +
provider);
        }

        return (String) response.getData().get("api_key");
    }

    @Override
    public void rotateSecret(String provider, String newSecret) {
        Map<String, Object> data = Map.of("api_key", newSecret);
        vaultTemplate.write("secret/signature-router/" + provider, data);
    }
}
```

7.5 TLS Configuration

```
server:
  ssl:
    enabled: true
    key-store: classpath:keystore.p12
    key-store-password: ${KEYSTORE_PASSWORD}
    key-store-type: PKCS12
    key-alias: signature-router

  # HTTP/2 support
  http2:
    enabled: true
```

7.6 Rate Limiting per Customer

```
@Component
public class CustomerRateLimiter {

    private final RateLimiterRegistry registry;

    public void checkRateLimit(String customerId) {
        RateLimiter limiter = registry.rateLimiter(customerId,
RateLimiterConfig.custom()
            .limitForPeriod(10)          // 10 signatures
            .limitRefreshPeriod(Duration.ofMinutes(1))  // per minute
```

```
            .timeoutDuration(Duration.ZERO)  // fail immediately
            .build());

        if (!limiter.acquirePermission()) {
            throw new RateLimitExceededException(
                "Customer " + customerId + " exceeded rate limit (10/min)"
            );
        }
    }
}
```

7.7 Input Validation

```
@RestController
@Validated
public class SignatureController {

    @PostMapping
    public ResponseEntity<SignatureResponse> createSignature(
        @RequestHeader("Idempotency-Key") @Pattern(regexp = UUID_REGEX) String
idempotencyKey,
        @Valid @RequestBody CreateSignatureRequest request
    ) {
        // Validation annotations in DTO
        // Additional business validation in use case

        SignatureResponse response = startUseCase.start(request);
        return ResponseEntity.status(CREATED).body(response);
    }
}

@Data
public class CreateSignatureRequest {

    @NotNull(message = "customerId is required")
    @Size(min = 16, max = 255, message = "customerId must be pseudonymized (16-255
chars)")
    @Pattern(regexp = "^CUST_[A-Za-z0-9_-]+$", message = "Invalid customerId
format")
    private String customerId;

    @NotNull(message = "transactionContext is required")
    @Valid
    private TransactionContext transactionContext;
}
```

7.8 SQL Injection Prevention

```
// JPA with parameterized queries (built-in protection)
@Repository
public interface SignatureRequestJpaRepository extends
JpaRepository<SignatureRequestEntity, UUID> {

    @Query("SELECT s FROM SignatureRequestEntity s WHERE s.customerId =
:customerId")
    List<SignatureRequestEntity> findByCustomerId(@Param("customerId") String
customerId);

    // NEVER use native queries with string concatenation
}
```

7.9 Audit Logging

```
@Aspect
@Component
public class AuditAspect {

    private final AuditLogRepository auditRepo;

    @AfterReturning(
        pointcut =
"@annotation(com.bank.signature.infrastructure.audit.Auditable)",
        returning = "result"
    )
    public void logAudit(JoinPoint joinPoint, Object result) {
        Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
        String actor = auth != null ? auth.getName() : "SYSTEM";

        MethodSignature signature = (MethodSignature) joinPoint.getSignature();
        Auditable auditable =
signature.getMethod().getAnnotation(Auditable.class);

        AuditLog log = AuditLog.builder()
            .id(UuidCreator.getTimeOrderedEpoch())
            .entityType(auditable.entityType())
            .entityId(extractEntityId(joinPoint))
            .action(auditable.action())
            .actor(actor)
            .changes(buildChanges(joinPoint.getArgs(), result))
            .ipAddress(getClientIp())
            .userAgent(getUserAgent())
            .createdAt(Instant.now())
```

```
            .build();

        auditRepo.save(log);
    }
}

// Usage
@Auditable(entityType = "RoutingRule", action = "CREATE")
public RoutingRuleResponse createRule(CreateRuleRequest request) {
    // ...
}
```

---

8. Security Scanning

8.1 Dependency Scanning

```
<!-- pom.xml -->
<build>
    <plugins>
        <plugin>
            <groupId>org.owasp</groupId>
            <artifactId>dependency-check-maven</artifactId>
            <version>8.4.0</version>
            <executions>
                <execution>
                    <goals>
                        <goal>check</goal>
                    </goals>
                </execution>
            </executions>
            <configuration>
                <failBuildOnCVSS>7</failBuildOnCVSS>
            </configuration>
        </plugin>
    </plugins>
</build>
```

8.2 Container Scanning

```
# .github/workflows/security.yml
name: Security Scan

on: [push, pull_request]

jobs:
  scan:
```

```
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Run Trivy vulnerability scanner
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: signature-router:latest
          format: 'sarif'
          output: 'trivy-results.sarif'
          severity: 'CRITICAL,HIGH'
```

Status: ✅ **COMPLETE - OBSERVABILITY & SECURITY DOCUMENTED**

Next Steps:

- Set up Prometheus + Grafana
- Configure distributed tracing (Jaeger/Zipkin)
- Integrate Vault for secrets
- Implement PII pseudonymization
- Configure TLS certificates